EE 332 Real Time Systems
Midterm Examination

Wednesday February 12, 2003

General:

- Two hours (2:30pm to 4:30pm)
- Open book and open notes

1. 15 Marks (3 marks each)

a) Briefly describe the difference between call-by-value and call-by-reference and give an example of each.

b) Some processors provide an instruction that allows the processor to go to sleep until an interrupt occurs. Why is this desirable and useful?

c) Is the following routine thread-safe? Provide a "yes", "no", or "cannot determine" answer and give a brief justification for your answer:

```
int     GlobalCount;

void IncGlobalCount( void )
{
   GlobalCount++;
}
```

d) Show the Moore Finite State Automata for a system that scans for the bit sequence 01111110 in a string of bits (the sequence is allowed to appear anywhere in the bit string).

e) Why is it important for a real-time system to have a watchdog timer? Is a timer interrupt routine an appropriate point in the software to reset the watchdog timer, why or why not?

2. 20 Marks (10 + 10)

Fibonacci numbers follow the sequence 1, 1, 2, 3, 5, 8, ... Except for the first two numbers, each number is the sum of the preceding two numbers. A recursive routine for determining a particular number in the sequence is given by:

```
int fibonacci( int number )
{
   if( number < 2 )
   {
      return 1;
   }
   else
   {
      return( fibonacci( number - 1 ) + fibonacci( number - 2 ) );
   }
}
```

a) Show how the fibonacci recursive algorithm works by showing the values of the parameters to each call of fibonacci( ), how many times fibonacci( ) is called, and the return value of each fibonacci( ) when you call "fibonacci(5)".

b) Given the 8086 assembler code shown on the next page generated from the fibonacci C code, show the maximum length call stack when you call "fibonacci(5)". Indicate the value and what is represented by each word of the stack.

16-14 = 20
15-14 = 19
E-14 13

```
_fibonacci
        push    bp
        mov     bp,sp
        cmp     word ptr [bp+4],2
        jge     short @1@86
        mov     ax,1
        jmp'    short @1@114
@1@86:
        mov     ax,word ptr [bp+4]
        dec     ax
        push    ax
        call    near ptr _fibonacci
        pop     cx
        push    ax
        mov     ax,word ptr [bp+4]
        add     ax,-2
        push    ax
        call    near ptr _fibonacci
        pop     cx
        pop     dx
        add     dx,ax
        mov     ax,dx
@1@114:
        pop     bp
        ret
```

3. 25 Marks (10 + 10 + 5)

Consider a hypothetical microprocessor called the CPU332. The CPU332 has the following properties:

- 16-bit microprocessor (16-bit data bus, registers, etc.)
- One 16-bit accumulator ACC
- Four 16-bit general purpose registers R0, R1, R2, R3
- A 16-bit stack pointer SP
- A 16-bit flag word FLAGS. Bit 0 (Least Significant Bit) is the "Interrupt Enable" bit, a value of "1" enables interrupts, a value of "0" disables interrupts.
- A "push" instruction can push any register (ACC, R0-R3, FLAGS) onto the stack. It does this by first writing the value and then incrementing the value of SP by two.
- A "pop" instruction can pop any register (ACC, R0-R3, FLAGS) from the stack. It does this by first decrementing the value of SP by two and then reading the value.

a) Show the "pseudo assembler" for a "Yield( )" function for the CPU332. The pseudo assembler should show the order of pushes, pops, and saving and restoring of stack pointers. Indicate which registers are pushed and in what order. The code to determine the next task to run can be glossed over (but you should still indicate where in your Yield( ) function this occurs). Note that this Yield( ) function should be usable for either a cooperatively or preemptive multitasking system.

b) For your Yield( ) function from (a), show the organization, addresses and values of the initial stack for a Task whose entry address is 2234H and whose stack space starts at 1000H. Remember to show required initial values for any values on the stack (or XXXX for don't cares). What is the value of the initial stack pointer?

c) If the interface of created tasks is required to be:

```
void Task( short taskNum, char* taskName );
```

`taskNum` is a 16-bit integer indicating the number of this task.

`taskName` is a 16-bit pointer to a string which is the name of this task.

Show how the initial stack is now different from part (b) as you now need to pass these parameters to the initial entry of Task. Do not worry about the value for these two parameters but do show where on the initial stack they reside (Hint: Remember C pushes parameters from right to left, rightmost parameter is pushed first, leftmost parameter is pushed last). What is the value of the initial stack pointer now?